Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2003-11

THE SYNCHRONIZED VEHICLE DISPATCHING PROBLEM

by

Louis-Martin Rousseau
Michel Gendreau
Gilles Pesant

April 2003

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

103-07102/

PUBLICATION
CRT-2003-11

THE SYNCHRONIZED VEHICLE DISPATCHING PROBLEM

by

Louis-Martin Rousseau[1]
Michel Gendreau[1,2]
Gilles Pesant[1, 3]

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds québécois de la recherche sur la nature et les technologies (FQRNT).

1. Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 (louism,michelg,pesant@crt.umontreal.ca)
2. Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7
3. Département de génie informatique, École Polytechnique de Montréal, C.P. 6079, succursale Centre-Ville, Montréal, Canada H3C 3A7

Département d'informatique et de recherche opérationnelle - Publication #1236

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2003

Abstract

This paper addresses a particular class of the Real Time Vehicle Dispatching Problem where there is need to service some customers with multiple resources subject to synchronization constraints. Real world applications are presented and a flexible solution method, based on Constraint Programming, is proposed. In order to facilitate future comparisons, benchmark problems are derived from well-known vehicle routing benchmark problems and a transformation method is detailed. Results on these problems show that solutions are sensitive to the number of synchronization constraints as well as the requests arrival rate.

Keywords: real time vehicle dispatching problem, synchronization constraint, constraint programming.

Résumé

Cet article présente un cas de problème de répartition de véhicule en temps réel où certains client doivent être visités par plus d'un véhicule et où ces visites doivent être synchronisées. Nous présentons des applications réelles de ces problèmes et nous proposons une méthode flexible basée sur la programmation par contraintes. Afin de faciliter les comparaisons, nous introduisons aussi un ensemble de problèmes tests dérivés des problèmes bien connus de Solomon. Des résultats sont donnés pour ces problèmes et la sensibilité du système face au nombre de contraintes de synchronisations est étudiée.

Mots-clés: répartition de véhicule en temps réel, contraintes de synchronisation, programmation par contraintes.

Introduction

The synchronized Vehicle Dispatching Problem (SVDP) is a transportation problem which arises in many real world situations. One which illustrates well this problem is a special version of the Dial-a-Ride problem, in which disabled persons require assistance in order to prepare for transportation. In this case, a special team is sent to the residence of the customer a few minutes before the vehicle to insure that the transfer is made safely and efficiently. The help can vary from dressing for winter conditions to providing wheelchair assistance and is not usually required at all times or by all customers. This means that the schedule of the special assistance teams needs to be synchronized with the schedules of the rest of the fleet.

Another example is taken from public service companies that offer different levels of service. The case of the cable companies providing internet services illustrates this problem well. When a customer subscribes to a high speed internet connection, some provider will offer a different installation package: a basic cable modem installation or a more complete software configuration package. The latter installation is usually done by two different technicians and their visits must be synchronized. In fact the company has to ensure that the hardware technician will always precede the software technician and that their visit will be as close to one another as possible.

This problem also occurs in large hospitals where patients are transferred between buildings. The dispatcher who plans the transfers for a given period must ensure that the orderlies of both buildings are synchronized with the ambulance.

To our knowledge this problem has not yet been presented in the literature, however there are many variants of the Vehicle Dispatching Problem covering many real life applications. Gendreau and Potvin [4], Qiu and Hsu [15](in the case of Automated Guided Vehicles) and Psaraftis [14] have published detailed surveys describing the different problems and solution approaches. Most efficient algorithms address this problem with local search techniques (like Tabu search for instances). Ichoua [6] has later introduced a taxonomy to further categorize the problems of this class.

The next section presents a model for the SDVP while section 3 describes a solution method to solve this model. In section 4, a transformation is given to produce benchmark problems from known Vehicle Routing instances and results on those problems are given in section 5.

1 Problem Description

This section presents a Constraint Programming model for the SVDP. The basic problem that we have chosen is the Real Time Vehicle Dispatching with Time Windows and Capacity Constraints which is here referred to as the Vehicle Dispatching Problem (VDP). This problem can be described as follows: given a set of customers C, a set of vehicles V, and a depot d, find a set of routes starting and ending at d, such that a maximum number of customers in C are

visited by exactly one vehicle. Each customer having a specific demand, there are capacity constraints on the load that can be carried by a vehicle and each customer c must be visited after time a_c and before time b_c. One can wait in case of early arrival, but late arrival is not permitted. When a customer calls for service the dispatcher must be able to tell the customer in a very short amount of time whether he will be served or not. Some customers ask for service one or more days in advance and some call during operational hours.

In the synchronized version of this problem, customers can ask for special services. They will be referred as *special* customers. Such customers will need to be served, in addition to the regular vehicle, by a *special* vehicle. Synchronization requirements such as precedence constraints and restriction on the time delay between regular and special visits are also specified. Capacity constraints are, however, not defined for special vehicles since they usually perform services and not pickups nor deliveries.

The synchronization constraints make this problem quite hard to solve using traditional local search methods. For instance, if the insertion of a special customer delays the route of a special vehicle then all regular visits associated with that route must also be delayed, independently of the vehicle that served them. But delaying a regular vehicle means delaying also the visits associated with other special customers and so on. This high number of interconnections means that to insert one customer one might have to recompute the visit time of every customer already inserted. Constraint Programming thus seems well suited for this kind of problem since it not only provides an easy way to express the synchronization constraints but, by representing visit times as domain variables and by propagating the synchronization constraints only when needed, it allows an efficient implementation of those constraints.

1.1 Constraint Programming

Before presenting the model, a brief description of the Constraint Programming paradigm is presented here. This approach, which has been very successful in solving hard combinatorial problems in the field of scheduling, planning, and transportation ([1], [19], [20]) has been the subject of a textbook [11] and survey [8].

Traditionally a Constraint Programming model is composed of a set of variables (X), a set of Domains (D), and a set of constraints (C) specifying which assignment of values in D to variable X are legal. The efficiency of the Constraint Programming paradigm lies in powerful constraint propagation algorithms which remove from the domain of the variables the values which will generate infeasible solutions. If propagation does not suffice in finding a feasible solution then a branching process is necessary to further narrow the domains; a feasible solution has been found when each domain contains only one value.

The branching process is thus useful to search for solutions when propagation is not sufficient, as it is the case in most difficult combinatorial problems. Typically, at each node of the search tree the following four steps are taken: first an unfixed variable is selected, then a value of its domain is chosen. The

selected variable is fixed to the chosen value, and constraint propagation occurs. If during propagation the domain of a variable is emptied then the solver has detected an inconsistency in the previously taken decisions and the whole search process backtracks, typically by choosing another value for the variable. When propagation terminates while there are still some unfixed variable, then the solver creates a new search node and goes on with the procedure just detailed. The branching strategy is thus defined by *variable* and *value* selection policies. This is the most simple and frequently used branching strategy but more intricate policies are often used. For instance one could split the domain of a variable into to two sets of similar size or even use two opposing constraints to define two branching directions.

It is fairly simple to extend this method to solve combinatorial optimization problems, that is to identify the feasible solution which minimizes (or maximizes) a given objective function. Once a feasible solution as been identified, the set C is extended to contain a new constraint specifying that future feasible solutions should have a strictly better cost than the cost of the solution just identified. The solver will thus keep searching for solutions of improving quality until it can prove that the last one it found was the optimal solution.

1.2 Model

Let N be the set of all customers. The depot is copied $2V$ times, where V is the number of vehicles, so that each route starts and ends at a different depot. Let then I and F be respectively the set of initial and final depots. Each special customer c is represented by two nodes, the regular node c and $c^\star$ the special node associated to c. The following model (M) is used to represent the problem:

1.2.1 Parameters

d_{ij}	Distance from node i to node j.
t_{ij}	Travel time from node i to node j.
l_i	Load to take at node i.
s_i	Service time at node i.
a_i, b_i	Bounds on node i's time window.
$a_i^\star, b_i^\star$	Synchronisation window of customer i.
K	Capacity of the vehicles.
C_r	Set of regular nodes.
C_s	Set of special nodes.
V_r	Set of regular vehicles.
V_s	Set of special vehicles.

1.2.2 Variables

$S_i \in C_r \cup F$	$\forall i \in C_r \cup I$	Successor of regular node i.
$S_i \in C_s \cup F$	$\forall i \in C_s \cup I$	Successor of special node i.
$V_i \in V_r$	$\forall i \in C_r$	Vehicle servicing regular node i.
$V_i \in V_s$	$\forall i \in C_s$	Vehicle servicing special node i.
$T_i \in [a_i, b_i]$	$\forall i \in C_r \cup I \cup F$	Time of visit of regular node i.
$T_i \in [a_i - a_i^*, b_i + b_i^*]$	$\forall i \in C_s$	Time of visit of special node i.
$L_i \in [0, K]$	$\forall i \in C_r \cup I \cup F$	Truck load at regular node i.

1.2.3 Constraints

$AllDifferent(S)$		(1) Conservation of flow.
$NoSubTour(S)$		(2) SubTour elimination constraint.
$S_i = j \Rightarrow L_i + l_i = L_j$	$\forall i \in C_r$	(3) Capacity constraints.
$S_i = j \Rightarrow T_i + s_i + t_{ij} \leq T_j$	$\forall i \in C_r$	(4) Time window constraints.
$S_i = j \Rightarrow V_i = V_j$	$\forall i \in N$	(5) Vehicle identification.
$S_i = i \Leftrightarrow S_{i^*} = i^*$	$\forall i^* \in C_s$	(6) Regular and special node association.
$T_i - o_i^* \leq T_{i^*} \leq T_i + b_i^*$	$\forall i^* \in C_s$	(7) Time windows synchronization.

1.2.4 Objective

$min \sum_{i \in N} d_{iS_i}$	Minimizing the total distance travelled

The *AllDifferent* (1) constraint is used to express conservation of flow in the network. The nature of the decision variable already enforces that each node has exactly one outgoing arc, but we also need to make sure it also has exactly one ingoing arc. To do so it necessary to insure that no two nodes have the same successor, which is the role of the *AllDifferent* constraint. This property is obtained by solving a matching problem in a bipartite graph and by maintaining arc consistency in an incremental fashion as described in [16].

The *NoSubTour* constraint (2) is taken from the work of Pesant *et al.* [12]. For each chain of customers, the first and last visits are stored and when two chains are joined together (when a variable is fixed and a new arc is introduced), two actions are taken. First, the information concerning the first and last visits of the new (larger) chain are updated, and then, the value of the first customer is removed from the domain of the S variable of the last customer.

Constraints sets (3) and (4) enforce that time windows and capacity constraints are respected at regular nodes, while (5) links the vehicle variables V to the successor variable S. We also introduce constraint set (6) to make sure that the regular and the special nodes of a special customer must be both visited or rejected together. Constraint set (7) is the synchronization constraints which insures that for each special customer the visit the special node is coordinated with the visit of the regular node.

The objective in this problem could be any problem-specific function that represents some goal to achieve. In this model, the objective function is defined as the minimization of travel cost even if the real objective is to maximize the

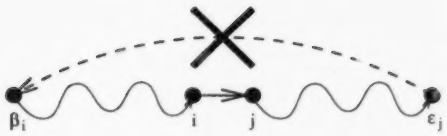

Figure 1: NoSubTour constraint

number of the served customers. This is, in fact, due to the solution method proposed which is based on the successive insertion of customers as they request service. Since customers are added one at a time an objective function trying to augment the number of visited customers would serve no purpose as it would only affect the one customer being inserted. Whereas the minimization of travel distance generates more space in the routes and thus facilitates future insertions. The next section will describe this insertion process and show how new customer insertion is enforced.

2 Solution Method

Building an exact solution method for this SVDP is very hard since the dynamic and real time components prevent any solution method from having all the information and decision flexibility at the same time. In the early stages of the dispatching process, the solution method does not possess the list of all the customers who will require service and thus cannot perform optimal planning. Whereas at the end of the process, when most of the information is known, vehicles are already on the road and have served a good number of customers. These decisions cannot be undone and directly impact the ability to service new requesting customers. Moreover, calling customers need to be told whether they will be served or not in a very short amount of time (typically a few seconds), which further limits the possibility of building optimal solutions.

The solution method proposed in this paper (and illustrated in figure 2) relies on the successive insertion of customers as they request service, while local search methods are applied between requests. This kind of procedure is very straightforward and has been applied in numerous real time dispatching algorithms. The first methods to solve real time dispatching problems were proposed in the 70's by Wilson and his colleagues ([22], [23], [21]) in the context of a demand-responsive transportation system in Haddonfield, NJ. These methods used simple insertion to incorporate each new customer into the current solution. This scheme was later applied ([17], [10]) to disabled person transportation systems when some of the requests were known in advance. The metaheuristics proposed in [13], [3], and [2] were developed for the static version of the problem and adapted to deal with the dynamic context. They essentially perform local search until some event (usually a new request) stops the search and triggers the insertion operator.

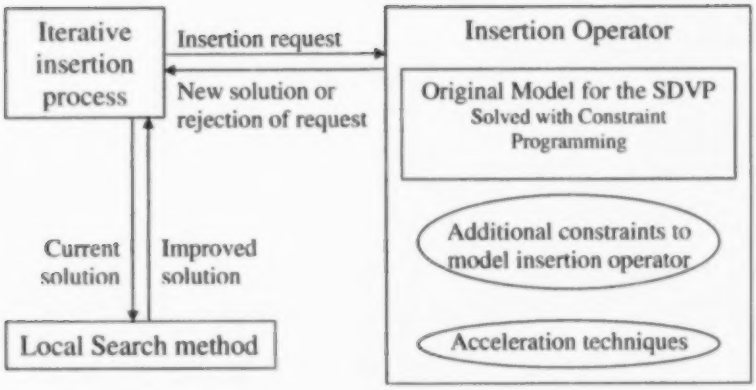

Figure 2: Overall method

2.1 Insertion Procedure

There are many possibilities for insertion strategies: insertion in the first possible positions and insertion in the best possible positions are two of the most popular ones. Implementing these operators in Constraint Programming allows easy handling of complex constraints such as synchronization. The rest of this section will detail the implementation of the insertion operators using the model previously presented and give insight into the local search used to improve solution quality.

2.1.1 Constraint-Based Insertion

The insertion operator can be modelled using Constraint Programming. Starting from a current solution (visiting some of the customers who have previously requested service) and given that S' is a copy of the values of S of the current solution, that the set of previously inserted (special) customers is defined by P ($P^\bullet$), and that the customer to insert is given by the index $k(k^\bullet)$, performing the insertion is then only a matter of solving the model (M) with the additional following constraints.

$S_k \in P \cup F$		Node k must precede an inserted customer.
$S_i \in \{S'_i, k\}$	$\forall i \in C_r \backslash \{k\} \cup I$	Only simple insertions are permitted.

And the equivalent if k is a special customer.

$S_{k^\bullet} \in P^\bullet \cup F$		Node $k^\bullet$ must precede an inserted customer.
$S_i \in \{S'_i, k^\bullet\}$	$\forall i \in C_s \backslash \{k^\bullet\} \cup I$	Add only if k needs special visit.

When customer k's request becomes known, most vehicles are on the road and some visits have already been performed. Since the previous constraints do not deal with the real time dimension of the SDVP, it is necessary to add two additional sets of constraints. The first set enforces that, at the current time denoted w, every customer that has not already been visited should be visited later than w. The second one forbids insertion after a customer when the vehicle serving it has already left the customer location[1] . By letting the latest departure time[2] of each customer i be denoted by d_i, the following update can be performed before the insertion search is started.

$$d_i > w \rightarrow T_i \geq w \quad \forall i \in N$$
$$d_i \leq w \rightarrow S_i = S'_i \quad \forall i \in N$$

Finding a feasible solution to the problem defined by model (M) enriched of these additional constraints gives a *first insertion* solution servicing customer k, while solving the problem to optimality yields the *best insertion* of k. It is obvious that the best insertion policy should provide better results since it explores a larger space of solutions, although once combined with a local search procedure (see section 2.2) it is not clear whether this advantage remains. The results section of the paper will compare both insertion strategies. Finally, if no solution to the modified (M) problem exists then the new customer cannot be inserted and his request will be rejected.

2.1.2 Acceleration Techniques

The model previously defined can easily tackle all small and medium size problems, ranging form 100 to 400 customers, but it significantly slow downs when the number of customers exceeds 400. In order to accelerate the method so that it can solve problems with up to 1000 requests, further filtering techniques are introduced.

The first one is a redundant constraint taken from the work of Pesant *et al.* [12] that allows to detect inconsistencies in the time windows earlier in the search tree. This constraint, which is checked every time the bounds of one of the T variables is modified, stipulates that :

$$min(T_i) + s_i + t_{ij} > max(T_j) \quad \Rightarrow S_i \neq j \quad \forall\, i, j \in N$$

One can also note that the bounds on the domains of the resource variables (T and L) of the current solution are valid for the solution obtained after a new customer is inserted. For instance, the latest time of visit of each customer cannot be extended after an insertion occurs. This means that the bounds obtained in a given solution can be applied directly in all following insertion problems. From this observation another filtering algorithm can be derived. Given that

[1] Ichoua [6] has proposed to diverge enroute vehicles in order to serve a new customer.

[2] This information can be problem dependent, but it is defined here as $min(b_i, b_{S_i} - t_{iS_i})$.

T' and L' are copies of the bounds of the T and L variables of the last solution, the following constraints can be added to the model.

$$min(T'_i) \leq T_i \leq max(T'_i) \quad \forall i \in N$$
$$min(L'_i) \leq L_i \leq max(L'_i) \quad \forall i \in N$$

Since capacity is additive in a very straightforward manner (as opposed to the time dimension which can suffer waiting delays), it can also be checked before the insertion process begins. This yields another set of constraints which accelerate the search process, where k is again given as the index of the node to insert.

$$l_k + min(L'_j) > K \quad \Rightarrow V_k \neq j \quad \forall j \in F$$

The variable selection policies traditionally used to solve this kind of problem are based on a *first fail* strategy, which means that the solver tries to branch on the most constrained variable first (usually the one with the smallest domain) in order to detect inconsistencies as early as possible in the search tree. However this strategy is not very effective in the context of the proposed insertion strategy because, after all the insertion constraints have been introduced, the domains of all (but one) S variables contain only two values (the previous successor and the customer to be inserted). The only S variable that contains more values is the one associated with the new request. Fixing this variable is equivalent to choosing the insertion position and for this reason S_k is always chosen as the first variable of the branching process. As for values, they are selected in a variable's domain on a most promising basis, meaning the value associated with the minimum cost.

2.2 Local Search

The insertion process is a greedy procedure in the sense that it does not reconsider the relative order of the inserted customers nor the vehicle they have been assigned to. To overcome this limitation, the dispatching system can make use of the idle time between requests of service to improve the current planning. This improvement phase is achieved with local search, where small modification are made on the current solution in an attempt to improve its quality.

The local search process is defined using a set of operators and a policy on how to use them (metaheuristic). The set of selected operators are the standard ones used in vehicle routing applications (*2-Opt*, *Or-Opt*, *Cross*, *Exchange* and *Relocate*). These operators are used in conjunction with two different metaheuristics: a simple descent strategy and a more complex Guided Local Search (GLS) [9]. Guided Local Search, like tabu search is allowed to visit solution of decreasing quality to avoid local minimums. GLS penalizes certain moves which tend to revisit already explored region of the search space, just like tabu search

which forbids those moves. All the operators and metaheuristics are taken from the ILOG Dispatcher library [7].

Since the local search procedure can only be used between customer calls for service, the amount of time spent improving the solution is not known *a priori*. Therefore the guided local search metaheuristic, which like tabu search can run continually until a new request comes, is more appropriate in a real world situation because it uses the maximum amount of time available for improvement. In a simulation context however, a descent strategy which stops after reaching a local minimum has the advantage of having performances which are, to a certain point, independent from simulation time. Again, the results section provides a performance comparison for both approaches.

3 Experimental Results

This section presents benchmark problems developed from the Synchronized Vehicle Dispatching Problem which are used to compare different solution methods parameter settings.

4 Benchmarking Problems

The static Vehicle Routing Problem with Time Windows (VRPTW) is probably one of the most benchmarked optimization problems. In 1987 Solomon proposed a set of problems [18] which were to become the reference VRPTW benchmarks.

These problems are grouped in three sets: the geographical data are randomly generated in problem sets R, clustered in problem sets C, and a mix of random and clustered structures in problem sets RC. The customer coordinates are identical for all problems within one type (i.e., R, C and RC). The problems differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows which are hardly constraining. These 100 customer problems have later been extended by Homberger and Gehring [5] to larger instances with up to 1000 customers. This set of problems, even if it is not representative of real life instances, facilitates the comparison between methods that address similar problems.

In the dynamic Vehicle Routing Problem field, most researchers have built problem generators that construct instances according to the probability distributions which best represent the dynamic contexts that their method addressed. This yields test problems similar to those that could be encountered in the real world but makes comparisons between methods much more difficult.

To facilitate further research on the SDVP, a set of dynamic benchmark problems is derived from the well-known Solomon problems. The transformation method which renders the static problems dynamic is first presented followed by the variant which creates the synchronized versions.

The Solomon problems describe a set of customers defined by integer identifier, a set of coordinates in a plane, a quantity of product requested, a time

window, and a service time. To generate dynamic problems it is only necessary to determine at which point in time each request becomes known to the dispatcher. Some instances may also show different levels of dynamism, meaning that a certain proportion of the requests are known in advance and that a pre-planning phase is possible before the real-time dispatching process begins. This is namely the case when customers call one or several days prior to their service day. Such instances are generated by selecting a number of customers which constitute a static instance of the VRPTW. This instance can be solved by any method and the best solution found is then used as the starting initial solution for the first customer insertion.

The dynamic problems are generated with the following parameters: customers whose time windows start at time 0 (those who can be served right away) are considered to be static customers. All other visits i are to be requested at time of a_i/α. For instance, α set to 2 means customers call for service at a time equivalent to half of their time window lower bound.

Once this is done, building a synchronized problem is just a matter of choosing which customers are going to be special, setting the number of special vehicles and giving values to special visit time windows (a^* and b^*). In the proposed transformation method, the special customers are distributed evenly across the dispatching horizon. The percentage of customers who will be considered special is given by β and the number of special vehicles is equal to β times the number of vehicles usually needed to solve the problem[3]. To select the special customers among those present in the original benchmark files, the following rules are applied:

$$(i \mod (100/\beta) > 0) \quad \Rightarrow \quad i \in C_r \ \forall i \in N$$
$$(i \mod (100/\beta) = 0) \quad \Rightarrow \quad i \in C_s \ \forall i \in N$$

4.1 Simulation

In order to simulate the arrival and dispatching process of a normal day of operations using the transformed benchmark problems, it is necessary to generate a mapping between the different time components specified in the test files (like time windows for instance) and the real time during the simulation process. Since the instances size varies greatly a fixed simulation time wouldn't yield representative results (i.e. smaller problems would be to easy). We thus need to compute a *per customer* simulation time. This mapping is achieved by defining γ, the average time interval between requests, and using T_{max}, the depot's closing time in the test file. A total runtime R is first computed as the product of γ by the total number of customers ($R = \gamma * |N|$), then a mapping factor $\mu = T_{max}/R$ is defined to link the two time dimensions. Once μ is defined, clock time can be converted to problem time by a simple multiplication.

[3]For the Homberger problems, these number were extrapolated from the solutions to the Solomon problems of size 100. For instance C.2.200.01 was said to need twice as many vehicles as C.2.100.01

Problem	First Insertion			Best Insertion		
Size	No LS	Desc.	GLS	No LS	Desc.	GLS
100	20.8	11.1	11.4	16.3	10.6	10.6
200	8.5	4.6	4.8	6.8	4.6	4.7
400	10.5	7.4	7.5	8.4	6.7	6.6
600	11.3	10.0	10.0	9.5	8.8	8.7
800	14.1	13.5	13.5	12.7	12.2	12.3
1000	16.6	16.4	16.4	17.6	17.5	17.5

Table 1: Percentage of rejected customers for the SDVP with $\alpha = 2$, $\beta = 10$ and $\gamma = 10$ sec.

4.2 Results

This section describes the performance of the proposed method on the Synchronized Vehicle Dispatching Problem. A set of synchronized problems was created by setting γ to 10 seconds and β to 10%, meaning one out of 10 customers is considered to be special. The value of (a^*, b^*) was set to (10,0) for each special customer so that the special visit could start up to 10 units of time before the regular visit. Table 1 reports the proportion of rejected customers against problem size using the proposed insertion operators and local search methods. There are 56 problems of size 100 and 60 problems of size 200,400,600,800 and 1000 for a total of 356 problems containing 185 600 requests.

Table 1 shows the rejection ratio yielded by the different solution methods when applied to synchronized problems. The main factor of performance for the smaller instances seems to be local search, whatever form it takes, since simple local search is as effective as Guided Local Search. For the larger problems however, *best insertions* outperforms local search. The combination of both technics yield even better results except for the problems of size 1000, where best insertions takes all the time available to evaluate a request and no local search can be applied. In the following tables the overall best combination (and the one most suited to real life cases) of GLS and *Best Insertion* will be used to measure the impact of the γ and β parameters, that is the rate of arrival of new requests and ratio of special customers.

But let first note the significant difference of rejection ratios between the 100 series and the rest of the instances. This discrepancy is probably due to fact that the instances of size 100 were generated by Solomon [18] while the larger problems were later proposed by Homberger [5] and thus present a different structure[4]. For the instances of size 200 to 1000, the rejection ratio increases with the size of the problem. This is explained by the fact that when instances become larger successive insertions become harder and the time available for insertion and local search becomes insufficient.

A key factor that influences the rejection ratio of customers is the average

[4]Although they were built in a similar fashion

Problem Size	Time interval between requests							
	$\gamma = 5$		$\gamma = 10$		$\gamma = 20$		$\gamma = 40$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	9.4	16.7	10.0	15.7	9.0	16.6	9.0	16.2
200	3.6	17.6	3.3	18.2	3.2	17.2	3.2	16.9
400	5.9	23.5	4.9	22.4	4.1	21.0	4.0	20.3
600	7.7	27.0	6.6	27.1	5.6	25.5	4.9	24.1
800	12.5	26.6	11.0	24.3	9.5	23.9	8.5	22.5
1000	16.4	29.1	16.0	30.5	15.0	30.9	12.6	28.6

Table 2: Percentage of rejected customers for the SDVP with $\alpha = 2$, $\beta = 10$

Problem Size	Percentage of special customers							
	$\beta = 5\%$		$\beta = 10\%$		$\beta = 25\%$		$\beta = 50\%$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	9.9	14.0	10.0	15.7	9.0	28.7	10.6	22.6
200	3.3	14.7	3.2	18.2	3.3	11.6	3.2	11.9
400	5.1	24.5	4.9	22.4	5.1	18.5	5.4	15.4
600	6.6	31.5	6.6	27.1	6.8	21.6	6.9	18.1
800	10.5	27.3	11.0	24.3	10.9	22.5	11.6	20.0
1000	15.4	33.5	16.0	30.5	18.1	29.1	23.1	23.9

Table 3: Percentage of rejected customers for the SDVP with $\alpha = 2$, $\gamma = 10$

time between requests. Table 2 clearly indicates that when more time is available to perform insertion and local search, more customers, both special and regular, can be served. It seems however that for smaller problems (100, 200), a time interval of 20 seconds is sufficient and that increasing the value of γ does not really improve solutions. Problems of size 400 and above would however probably benefit from increased time intervals.

Table 3 reports the rejection ratio according to the percentage of special customers. By looking at series 400 to 1000, we note that, although the percentage for regular customers are quite stable, the percentage of rejected special customers decreases when β increases. This is explained by the flexibility gained with the increased number special vehicles. Thus, when the number of special customers rises, more of these customers are rejected in absolute value, but it becomes relatively easier to insert them.

The conflicting results for 100 and 200 series are caused by the discrete nature of the number of special vehicles. In these instances there are too few regular vehicles to generate the proper ratio of special vehicles. This means that each instance either has too many or too few special vehicles in proportion to regular ones.

5 Conclusion

A new variation of dispatching vehicle problems was introduced to meet the specific need of a class of complex real world applications. The synchronization constraints could be applied in the context of any dynamic fleet assignment problem or even in the static case.

Constraint-based insertion operators were derived from the original model for the Synchronized Vehicle Dispatching Problem and were used to define a successive insertion procedure. Local search was also applied between requests to improve the current solution and facilitate future insertions. Results were given on a set of constructed benchmark problems to demonstrate the relative performance of the insertion operators and local search proposed, as well as the sensitivity to the rate of arrival of the requests and the ratio of special customers.

The problem was addressed using Constraint Programming for mainly two reasons. Firstly, the modelling power of this paradigm, which greatly eased the modelling phase, allows for a complete separation between the model and the search procedure. This feature makes it simple to add new constraints without having to modify the insertion operator or the local search method. Secondly but most importantly, the domain variable representation and the propagation techniques of CP eased up the handling of synchronization constraints.

References

[1] M. Dincbas, P. Van Hentenryck, and H. Simonis. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 8:75–93, 1990.

[2] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries. Publication CRT-98-10, Centre de recherche sur les transports, Université de Montréal, Montréal, 1998.

[3] M. Gendreau, F. Guertin, J.Y. Potvin, and E. Taillard. Tabu Search for Real-Time Vehicle Routing and Dispatching. *Tansportation Science*, 4(33):381–390, 1999.

[4] M. Gendreau and J.-Y. Potvin. Dynamic Vehicle Routing and Dispatching. *Fleet Management and Logistics*, pages 115–126, 1998.

[5] J. Homberger and H. Gehring. Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows. *INFOR*, 37:297–318, 1999.

[6] S. Ichoua. *Problème de gestion de flottes de véhicules en temps réel.* PhD thesis, Université de Montréal, 2001.

[7] ILOG S.A., 12, Avenue Raspail, BP7, 94251 Gentilly Cedex, France. *ILOG DISPATCHER*, 2001.

[8] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[9] P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with Time Windows. In *META-HEURISTICS Advances and Trends in Local Search Paradigms for Optimization*, pages 473–486. Kluwer Academic Publishers, 1999.

[10] O.B.G. Madsen, H.F. Ravn, and J.M. Rygaard. A Heuristic Algorihtm for a Dial-a-Ride Problem with Time Windows, Multiple Capacities and Mulptiple Objectives. *Annals of Operations Research*, 61:213–226, 1995.

[11] K. Mariott and P.J. Stuckey. *Programming with Constraints*. The MIT Press, 1998.

[12] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows. *Transportation Science*, 32:12–29, 1998.

[13] H.N. Psaraftis. Dynamic Vehicle Routing Problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 223–248. North-Holland, 1980.

[14] H.N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, (61):143–164, 1995.

[15] L. Qiu and W. Hsu. Scheduling and routing algorithms for agvs: a survey, 1999.

[16] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994.

[17] S. Roy, J.M. Rousseau, G. Lapalme, and J. Ferland. Routing and Scheduling of Transportation Services for the Disabled: Summary Report. Technical Report CRT-473A, Centre de Recherche sur les Transports, Université de Montréal, 1984.

[18] M. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–265, 1987.

[19] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Mass., 1989.

[20] M. Wallace. Practical Applications of Constraint Programming. *Constraints*, 1:139–168, 1996.

[21] N.H.M. Wilson and N.H. Colvin. Computer Control of Rochester Dial-a-Ride System. Technical Report R-77-30, Departement of Civil Engineering, Massachusetts Institute of Technology, 1977.

[22] N.H.M. Wilson, J.M. Sussman, H.K. Wang, and B.T. Higonnet. Scheduling Algorithms for Dial-a-Ride Systems. Technical Report USL TR-70-13, Urban Sustems Laboratory, Massachusetts Institute of Technology, 1971.

[23] N.H.M. Wilson and H. Weissberg. Adavance Dial-a-Ride Algorithms Research Project: Final Report. Technical Report TR-76-20, Departement of Civil Engineering, Massachusetts Institute of Technology, 1976.